on-hoard Manning for New Millennium Deep Space One Autonomy

Nicola Muscettola[1]
Chock Fry
Karma Rajan

Computational Sciences 1 Division

NASA Ames Research Center
Moffett Field CA 94035
{mus, chucko, karma}
@ptolemy.arc.nasa.gov

Ben Smith[2]
Steve Chien
Gregg Rabideau
David Yan

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive M/S 525-3660
Pasadena, CA 91109-8099
{smith, chien, rabideau, yan}
@aig.jpl.nasa.gov

Abstract- The Deep Space One (DS 1) mission, scheduled to fly in 1998, will be the first NASA spacecraft to feature an on-board planner. The planner is part of an artificial intelligence based control architecture that comprises the planner/schedulcr, a plan execution engine, and a model-basccl fault diagnosis and reconfiguration engine. This autonomy architecture reduces mission costs and increases mission quality by enabling high-level commanding, robust fault responses, and opportunistic responses to serendipitous events. This paper describes the on-board planning and scheduling component of the DS 1 autonomy architecture.

TABLE OF CONTENTS

1. INTRODUCTION

NASA's New Millennium Program (NMP) strives to achieve a "virtual presence" in space by deploying several spacecraft built "faster, better, and cheaper" than traditional spacecraft. Spacecraft autonomy is a crucial element in achieving this vision. It is expected that autonomous spacecraft will redu cc mission operations costs by taking over many of the operations that have typically been performed on the ground, and will improve mission quality by being more robust to failures and more responsive to unexpected opportunities than traditional spacecraft.

The first NMP mission Deep Space One (DS1), scheduled to launch in 1998 will feature an autonomy architect ure. The centerpiece of this architecture is the Remote Agent (RA), an artificial intelligence based

[1] Nicola Muscettola is under contract from Recom Technologies. Chuck Fry and Karma Rajan are under contract from Caelum Research.

[2] This paper describes work performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract from the National Aeronautics and Space Agency.

control system. The RA is derived from the NewMaap technology demonstration [1]. It consists of three components: the Planner/Scheduler (PS), the Executive (EXEC) [2], and a model-based Mode Identification and Recovery engine (MIR) [3]. This paper describes the Planner/Scheduler component of the Remote Agent.

The RA reduces mission costs and increases mission quality in several ways. First, it reduces mission costs by enabling high level commanding. This is a function both of the hierarchical nature of the RA architecture and the ability to generate plans on-board. Instead of painstakingly constructing detailed command sequences as is required in traditional missions, the ground can command the spacecraft with a handful of high-level goals. Changes to the mission plan are easily accommodated by changing the Seals. Transmission costs are also reduced, since goals take less time to transmit than their corresponding sequences.

Second, the RA reduces mission costs and improve mission quality by providing robust responses to failures that would normally require ground intervent ion. The PS provides responses that require the ability to look ahead and deliberate about global interactions, whereas the rest of the RA handles real-time failures requiring only local reasoning but quick reactions. The ability to respond both deliberative and reactively to failures provides far more robustness than traditional missions, and permits substantial savings in ground-operations resources that would otherwise be devoted to fail ure response. It also allows for lighter 1)SN (Deep Space Network the network of antennas used for deep space communication) coverage- roughly one pass per week since the spacecraft is less likely to be idle for several days after a failure, waiting for the ground to respond,

Finally, on-board planning can improve mission quality by taking advantage of fortuitous events, such as better than expected resource consumption. The same process used to "plan around" failures can be used to generate new plain that take advantage of a change in spacecraft state. This capability is important since it enables fundamentally new planetary exploration missions where round-trip light time (Jots not allow effective joy-sticking of the spacecraft from Earth.

Organization

The remainder of this paper is organized as follows. The DS1 mission is described by way of Context in Section 2, High level commanding is discussed in Section 3, along with examples Of mission goals and constraints. The PS is described in Section 4, the goal prioritization and reject ion mechanisms are described in Section 5 and the fault response mechanism is discussed in Sect ion 6 with examples from a 1)S 1 fault scenario. Opportunistic responses to fortuitous events are discussed in Section 7 and conclusions appear in Section 8.

2. THE MISSION

]n past missions, spacecraft have been fairly large and expensive (e.g. the Cassini mission to Saturn is budgeted for approx. $1 billion), and have used mostly older, established technologies in favor of newer, riskier ones. The spacecraft and the scientific data it was tasked with gathering were just too valuable to risk. As a result, many of the newer technologies have never bad a chance to fly, despite their potential advantages.

Enter the New Millennium program (NM])). Its objective is to develop new technologies and processes that wi 11 allow spacecraft to be built and flown "faster, better, and cheaper" than traditional missions. "1'here are six missions in the New Millennium program, of

which this one 1)eep Spat.c One *(1)S* 1) is the first.

The objective of the NM]' missions is not to do planetary science, but to validate new technologies in flight. The spacecraft arc relatively inexpensive (e. g. the DS1 Mission is capped al $138.5 million), and although there is a science component, its presence is primarily to stress the technologies. This shift of priorities allows the technologies to bc developed and validated without worrying about all the constraints imposed by typical science missions. If the ncw technologies prove worthy, they will bc used on future science missions.

The nominal mission is to fly by Asteroid 3352 McAuliffe in January 1999 and take a series of images, and then to repeat the process in a flyby for Comet West-Kohoutek-Ikemura in June 2000. Since one of the goals of spacecraft autonomy is to reduce ground operations costs, there will bc minimal ground support and very light DSN cover age throughout the mission only onc pass every two weeks.

DS1 bas thirteen ncw technologies aboard. During cruise, while the spacecraft is closing with the first encounter target, validation experiments will bc performed on each of the new technologies. Of these technologies, some arc mission critical. If they fail, the rest of the mission will bc seriously compromised or lost. These mission critical technologies will bc validated by the demands of the mission itself' as well as by specific validation tests.

The Remote Agent (f< A), as the control system for the spacecraft, is a critical mission technology. The RA consists of three components: the Planner/Scheduler (PS), tbc Executive (EXEC), and a model-based Mode Identification and Recovery engine (M IR).

The PS receives a set of high-lcvc] mission goals, either directly from tbc ground or as part of a pre-loaded *mission profile*, and generates a *plan* a set of synchronized procedures. Once executed, these commands will achieve the mission goals without violating resource, temporal, or safet y constraints. The EXEC receives tbc commands and ensures the correct dispatching of 10 W-1CVC1 commands to the real-time device drivers. MIR monitors device responses to commands, identifies possibly faulty components and suggests recovery actions to the EXEC.

Some of the other critical technologies arc the on-board optical navigator, the ion-propulsion engine (11'S), and the Miniature Integrated Camera Spectrometer (MICAS).

The on-board navigator determines t h e spacecraft trajectory and position based on images of the surrounding star field taken on a regular basis (every fcw days during cruise, and more often near encounter). The images arc taken with the MICAS camera, a new compact device with infrared (IR), ultraviolet (UV), and visible light sensors.

Unlike most missions, where velocity is accumulated with powerful chemical thrusters, DS1 uses an ion propulsion engines (IPS), which generates only a fcw millinewtons of force by ejecting energized xenon particles. By thrusting almost constantly, the required velocity can be achieved more efficiently than with chemical thrusters, albeit more slowly. The 11'S engine must bc shut down every fcw days to take images for optical navigation.

Over several months, tile spacecraft c1oscs with the encounter target body- asteroid or comet- and flies by at several kilometers pcr second, taking a sequence of MICAS images as it does so. The trajectory is updated over a f c w days based on increasingly frequent

images of the target. The flyby itself lasts perhaps 400 seconds, during which time the spacecraft must take a tightly timed sequence of images and make last minute course corrections.

3. High Level Commanding

A Remote Agent controller that includes an on-board PS enables a new approach to spacecraft commanding, *high-level commanding*. in this approach commands to a spacecraft take the form of abstract directives or *goals* instead of detailed streams of instructions. The responsibilities of PS are: (1) (o select among the proposed goals those to be achieved at any point in time; (2) to compromise between the level of achievement of the selected goals, and (3) to expand the *procedures* needed to achieve the goals. PS ensures the satisfaction of various synchronization constrains among procedures, and resolves resource conflicts. The Set of expanded procedures and constraints among them constitutes a *plan*.

In contrast, the traditional approach to spacecraft commanding is to develop a detailed sequence of time-tagged commands to the real-time device drivers. This extremely detailed level of commanding allows a high level of sequence optimization in order to "squeeze" as much performance as possible out of tbe spacecraft. 1 lowever, the drawback is that temporal and resource constraints and fault protection goals also have to be ensured at an extremely detailed level. The consequence is that developing a sequence is a very exacting and time consuming process, often requiring months of manual labor. Once generated, a sequence is very difficult to modify.

The primary benefits of hi?)l-level commanding when compared to traditional sequencing arc modularity, execution flexibility and robustness.

With respect to modularity, in Remote Agent the existence of an abstract plan makes very explicit the hierarchical decomposition of responsibilities between the different architectural components. This hierarchical approach can greatly reduce the amount of work needed to generate sequences. The procedures in a plan typically represent fairly complex sequences of instructions to the real-time device drivers. The expansion of this sequence is achieved by EXEC and MIR on the basis of the actual execution conditions. Since the plan already resolves synchronization and resource allocation constraints among procedures, the process of expanding an EXEC/MIR sequence is highly localized and, therefore, greatly simplified. Extensive validation of these small sequences is much simpler than the validation of sequences generated in the traditional approach. Localization of interactions among procedures and flexibility of procedure expansion at execution time has also the effect of making plan execution more robust to failures. This is discussed further in Section 6.

Execution flexibility depends on the fact that a plan is not simply a time-tagged sequence of commands, Procedures in a plan can be potentially executed in parallel. The plan explicitly represents and maintains temporal constraints between procedures. These derive either from the legal conditions under which the spacecraft hardware can be operated or from requests from ground operators. For example, a temporal constraint can express that procedure A must start from 30 to 60 minutes after procedure B, or that the execution of procedure B must occur while procedure C is in execution, or that procedure A ends exactly when procedure C starts. PS ensures the consistency of the network of temporal constraints in the plan and infers

time ranges during which a procedure can start and end. Unlike simple time tags, time ranges give EXEC the flexibility to compensate for execution delays caused by locally recoverable failures.

Relying on an on-board planner can also make fault protection simpler and more robust than traditional sequencing. in the traditional approach a sequence is infrequently uplinked to a spacecraft and therefore needs to include contingencies to handle a wide variety of failure conditions. When a major failure occurs execution of the single on-board sequence must be restarted, the sequence must command the assessment of the new execution conditions and react conditionally on the basis of this assessment. Because of the large number of possible failure conditions and the low level of the instructions in a sequence, the size of a robust sequence can be very large and the effort needed to build it very high. Plans are instead valid only for the execution conditions known at the time 1'S was invoked. For this reason the sequences eventually expanded from a plan are generally simpler and smaller. Fault protection goals, however, need not be compromised. When execut ion conditions differ so much from the initial assumptions that local failure recovery is insufficient, execution of the plan stops and 1'S is asked for a new plan that takes into account the new situation, Dealing with fault conditions on an as-needed basis simplifies the solution of the fault protection problem.

It has to be noted that some critical parts of the mission may still be so resource and time constrained to require optimization at the level of individual real-time instructions. in such cases high-level commanding allows several alternative ways to address the problem which are no worse than the traditional approaches. For example, the plan may simply include a single procedure for the entire critical sequence and EXEC will initiate execution of the canned sequence when encountering the procedure. Another possibility is to make each procedure correspond to an individual real-time command and let 1'S generate the sequence automatically. This may be as complex as the traditional approach and may not be effectively addressed by current automated planning technology. For DS 1, however, we will concentrate on demonstrating the modularity, flexibility and robustness of higi]-level commanding leaving advances in optimality to future missions.

4. GENERATING PLANS FROM GOALS

Figure 1 describes how the DS 1 on-board planner implements high-level commanding.

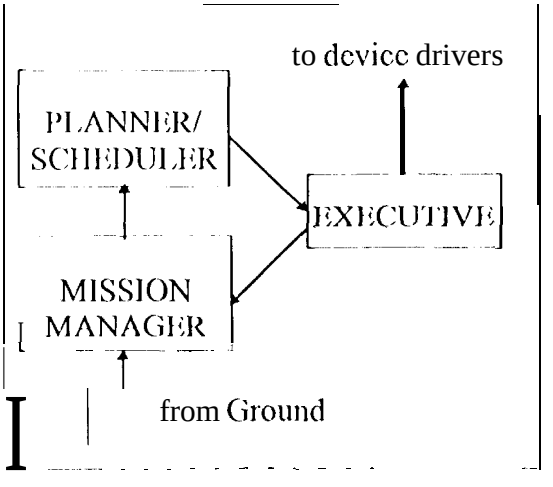


Figure 1: High Level Commanding

A long-term plan covering the entire mission, the *mission profile,* is stored and maintained on-board by the *Mission Manager (MM).* '1 he MM allows ground operations to modify mission goals by editing the mission profile. h4M also has the responsibility to respond to the EXEC's requests for new plans. When this happens MM selects a new set of goals from the mission profile, combines it with initial state information provided by the EXEC and sends it to the 1'S. The time horizon covered by PS is typically two weeks during cruise and

a few day during encounter. When a plan is ready, PS sends it to the EXEC. When EXEC has almost completed execution of its current plan, it sends a new request to MM; this also happens when the EXEC is maintaining the spacecraft in standby mode after the occurrence of a major failure.

Plan representation

Roth PS and MM represent plans using the same kind of data structure, the *plan database.* This is organized in several parallel *timelines*, *each* comprised of' a sequence of *tokens.* A timeline describes the future evolution of a single component of the spacecraft's state vector. The set of tokens active at a given point in time represent the value of the state vector at that time. Goals and procedures are both represented as tokens. Each token consists of a state variable descriptor (specifying to which timeline the token belongs), a type (a symbolic representation of the goal or procedure and its parameters), a start-time, an end-time and a duration.

For example, there may be one timeline describing the state of the engine (warming up, firing, or idle) and another describing the spacecraft attitude (pointing to a target, turning from target A to target B, etc.). Explicit temporal constraint synchronize tokens on separate timelines. For example, the spacecraft attitude must be pointing to target1 while the engine is firing. Temporal constraints can also enforce ordering of tokens on a single timeline (e.g., the engine must warm up for at least an hour before it fires). A plan involving these two timelines is shown in Figure 2.

Timelines can also represent the state of renewable resources, such as battery state of charge, non-renewable resources, such as fuel, and aggregat c rc.sources (i .e., resources that can be allocated in parallel to several consumers), such as electric power. '1 'emporal constraints synchronize resource al locat ion tokens with the corresponding consumer tokens.

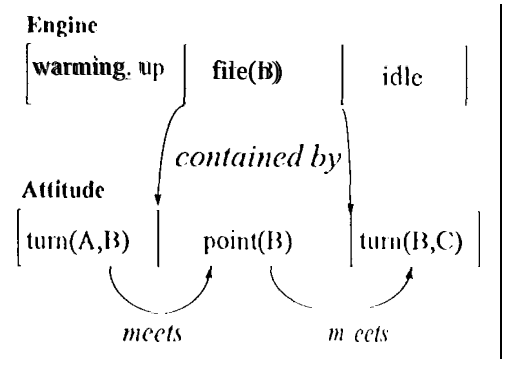


Figure 2: Example Timelines

The type of the resource tokens indicate the amount requested and the modality of consumption of the resource (e.g., constant, linear depletion). Resource timelines also include mechanisms to aggregate all parallel , requests, compare the cumulative requests with avai labi 1 it y and prevent resource over-usc (e.g., drawing more power than is available, or using more fuel than is allocated for the mission phase).

The plan database can represent a plan at any stage of partial completion, Incomplete plans can have gaps between tokens on a timeline. Also, an incomplete plan may include a request for a constraint between tokens (see the section on 'The Planning Model") that has not yet been implemented. 1'S will analyze the state of a database and add tokens and constraints until the plan is complete.

Wherever possible the plan database explicitly represents decision variables and constraints among them. The database uses constraint propagation mechanisms to infer valid ranges of values for variables (e.g., start or end times of tokens) and to detect inconsistencies(e.g.,

contradictory temporal constraints between tokens). This representation allows PS to concentrate on establishing constraints instead of selecting exact values for decision variables, an approach that often avoids over-commitment errors and therefore minimizes backtracking on commitments made earlier.

More representational details on the plan database can be found in [4].

The Planning Model

A valid plan must satisfy many constraints, including ordering constraints (e. g., the catalyst-bed heaters must warm up for ninety minutes before using the reaction control thrusters), synchronization constraints (e.g. the antenna must be pointed at the 1 Earth during uplink), safety constraints (e.g. do not point the radiators within twenty degrees of the sun), and resource constraints (e.g. the MICAS camera requires fifteen watts of power). These are all expressed as temporal constraints, or *compatibilities*, among tokens.

The planning model is a set of compatibilities that must be satisfied in every complete plan. More formally, a compatibility is a temporal relation that must hold between a *master token* and a /aI-get *tokens* whenever the master token appears in the plan. If the master token does not occur in the plan, the relation does not need to be satisfied.

A master token can have several compatibilities. '] 'hese are expressed as a Boolean expression of compatibilities called *compatibility trees*, as shown Figure 3. The tree in this figure says that the state in which the MICAS camera is on must be preceded by a state in which it is turning on, and followed by one in which it is turning off. While the camera is on, it consumes fifteen watts of power.

```
(Ml CAS_Ready)
:compatibilities
(AND
    (met_by (MICAS_Turning_On))
    (meets  (MICAS_Turning_Off) )
    (equal  (REQUEST (Power 15 )))
```

Figure 3: A Compatibility Tree

Whenever a MICAS_Ready token appears in the plan, it must be preceded by a MICAS Turning On token and followed by a MICAS_Turning_ Off token, and the power timeline is decremented by fifteen watts for the duration of the token (and must not go below zero available power for obvious reasons).

Initial State

The input to the 1'S is an initial state and a set of goals. The output is a *plan* that achieves the goals when executed from the initial state. For DS1 plans, the initial state of a plan is the first token on each timeline.

The initial state of the plan must match the state of the spacecraft at the time the plan is executed, Since it can take several hours to generate a plan (the baseline is eight hours), the initial state provided to the 1'S is necessarily a prediction of the future spacecraft state.

Under normal conditions, the prediction is made by the EXEC based on projection of the plan it is currently executing. Since the new plan will start at the end of the current plan, the projected initial state of the next plan is the final state of the current plan. "Iihe I'S also needs to know at what time the new plan should begin. This is also provided by the EXEC based on the earliest and latest end times of the current plan.

in off-nominal situations, there is no current plan from which to project the initial state of

the next plan. This happens when a non-recoverable failure occurs during execution of the current plan, or when the ground wants to upload a pre-defined (or "canned") plan.

In the case of a canned plan, the ground cannel easily predict the state of the spacecraft at the time the plan will be executed. The solution is to put the spacecraft into a known state in which it can persist until the plan is executed. This state is the initial state of the plan.

In the case of non-locally recoverable failures, the rest of the plan cannot be executed. in addition, the EXEC may have had to take actions not in the plan in order to get the spacecraft into a safe configuration following the failure. So the current plan cannot be used to predict the future state of the spacecraft. Fortunately, tbe safe configuration is a stable one. The exec uses this configuration as the initial state, and persists in this state until the new plan is ready.

1 f the plan was aborted due to a failed or degraded device, the failure must be noted as part of the initial state. For example, the II'S engine may be non-operational, or perhaps the MICAS camera is stuck on.

In tbe case of a stuck device, the token corresponding to the stuck state is asserted on the appropriate timeline over the entire planning horizon. This prevents the PS from generating a plan that requires the device to change states (the plan would fail). A non-functional device is declared unusable by placing a `not used` token at the start of the device's health timeline. The reason for saying the device is not usable rather than non-functional is so that the health of the device can be separated from the decision to use it.

For example, an intermittently failing device may be declared unusable by tbe EXEC or the ground. The EXEC and MIR can continue to track the device's health, while the EXEC, I'S and possibly the ground reason about whether to use it. Since tile PS only cares about usability, the tokens on the "health" timelines are `available` and not `used`.

A device can be available but degraded. If the 1'S needs to reason about the degraded modes, they are specified as arguments to the `available` token. For example, the battery capacity can be degraded even though the battery is still functional. The argument of the status token indicates the maximum charge level in amp hours (e.g., `available(24)`, `available(20)`).

Mission Manager

Mission operations commands the spacecraft through a plan database called the *mission profile.* On board tbe mission profile is maintained by a dedicated process, the *Mission Manager* (MM). DS1 will be launched with a mission profile for the entire mission. In principle this will allow the spacecraft to achieve the nominal mission without any additional uplink. in practice, M M provides mechanisms for ground operations to edit tbe mission profile and modify the mission goals while in flight.

The mission profile is an incomplete plan with its tokens representing what needs to be achieved by the mission. Unlike PS, MM dots not attempt to fill in gaps in the mission profile. Instead, when requested, M M determines the length of the next planning horizon, and selects the tokens that fall in the horizon and need to be sent to 1'S.

Figure *4 shows* a representative DS 1 mission profile. The `Waypoints` timeline contains a series of `waypoint ( . . . )` tokens, each representing a boundary point for a scheduling horizon. MM determines the length of the scheduling horizon by selecting the next

waypoint token such that there is enough time for J'S to produce a plan between the current time and the time of occurrence of the waypoint.

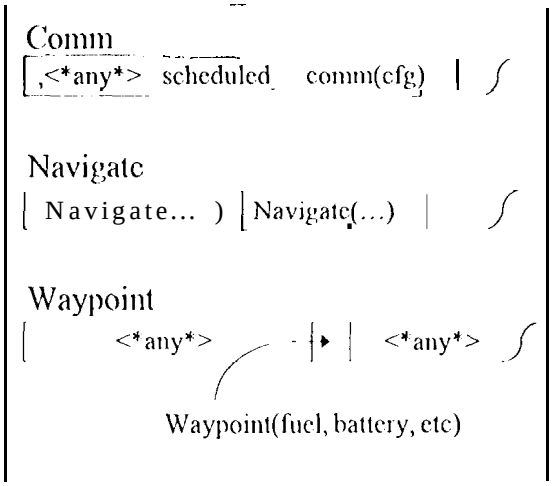


Figure 4: Goal Timelines

waypoint tokens also provide a set of check-Jmint conditions on resource usage that the plan must satisfy. These check-point conditions arc important to guarantee a well balanced achievement of all mission goals. Without them PS could be free to use a greedy approach and consume all available resources to maximize goal achievement within a few scheduling horizons. Since the PS' temporal perspective is limited, it is the mission designers' responsibility to provide long term perspective through waypoints.

The Comm timeline determines when the spacecraft is scheduled for communication with ground through a 1)SN pass. This is done by placing a schedu1edcomm token on the timeline, with start and end times corresponding to those of the pass. The argument of the token indicates the telecommunication configuration that the gi ound system is expecting. The Comm timeline is incomplete since it contains dummy C*any-value*> tokens within the goat tokens. This means that 1'S has the freedom to fill in the gap with whatever default procedures arc more convenient according to the domain model.

Other goal timelines arc specified similarly. For example, the navigation timeline has navigate tokens that indicate how often the on-board navigator should ask the spacecraft to take images of the star field.

Although the mission profile is design to express the entire nominal mission through a few timelines of goal tokens, sometimes ground may want to force the execution of special maneuvers. These can only be expressed through special networks of synchronized procedures. For this reason the mission profile includes also all timelines that usually contain procedure tokens expanded by 1'S. Ground has equal access to goal and procedure timelines and can therefore include tbc needed tokens in the mission profile. It is important to notice that since MM and PS make no a-priory distinction between goals and procedure tokens, ground can describe maneuvers only in part leaving to PS the responsibility to expand other procedures that may be needed to adapt to the actual execution conditions. These conditions are unknown to ground at the time of specification of the special maneuver.

On-board Goals

In addition to the goals in the mission profile, goals also conic from on-board systems, such as the navigator. This allows the spacecraft to modify its goals, and therefore its behavior, based on new knowledge that the ground may not yet have. This capability is particularly important since the spacecraft has only infrequent contact with the ground, and may have to act on the new knowledge before the next DSN pass.

This is especially true of navigation goals. At the beginning of each planning horizon, the PS asks the navigator what images should be taken and what course corrections are necessary based on images anti execution data from the previous horizon. The }'S then generates a plan that achieves these goals.

The goals must be generated on-board, since waiting for the next DSN pass to downlink the images and execution data, and then waiting for the ground to uplink navigation goals is not feasible. This is especially true near encounter, where several course corrections are made within a couple days, and some corrections are only a few hours apart.

Goals generated on-board are treated the same way as goals in the mission profile. However, they do raise sonic interesting issues. In particular, on-board goals may conflict with goals in the mission profile. 1 f the goals are mutually exclusive, then there is no plan that will satisfy all of the goals.

DS 1 addresses this problem in two ways. One is to prioritize the goals (see Section 5). *The* lower priority goals can be ignored, removing the conflict. The navigator's image goals have the lowest priority, since the navigator can still function adequately if it misses an occasional image. The second approach is to make sure that the goals are sufficiently flexible that there is always some way to satisfy all of them.

Planning Algorithm

The planner essentially searches in the space of incomplete or partial plans [5] with additional temporal reasoning mechanisms [6 and 4]. As with most causal planners, the I'S begins with a partial plan and attempts to expand it into a complete plan. The plan is complet c when it satisfies all of the compatibilities in the plan model, and all of the timelines have final tokens that end at or after the end of the plan horizon.

The unsatisfied compatibilities are also referred to as *open* compatibilities. An open compatibility is a temporal relation that must exist between a *master* token that is already in the plan and a *target* token that may or may not be in the plan. For example, the compatibility A meets B is open if A is in the plan, but B is not, or if A and B are in the plan, but the temporal relation is not explicitly en forced.

The I'S can satisfy an open compatibility in one of three ways. It can add the target token to the plan in such a way that it satisfies the temporal relation; it can adjust the start or end time of either the target or master token in order to satisfy the relation; or it can decide that the relation will be satisfied by a token in the next planning horizon, and can therefore be ignored. These options are called *adding, connecting*, and *deferring*, respectively. Deferred compatibilities are maintained in the plan, and carried forward to the next planning horizon as part of the initial state.

This basic loop is summarized in Figure 5, below. Each decision can be made non-deterministically, though in practice the decisions are guided by heuristics. If the wrong decisions is made, the 1'S will eventually reach a dead end and backtrack. It then tries one of the other decisions.

5. GOAL PRIORITIZATION

One of the most common problems when

While plan has open compatibilities:
1. pick an open compatibility
2. select and apply resolution strategy
3. if no resolution possible, backtrack.

Figure S: Planning Loop

developing a plan is the resolution of spacecraft resource over-slll>scril>tiolls. The problem stems from the fact that independent sources (e.g., tbc science team, the navigation team) compete for tbc usc of the limited on-board resources. The overall mission goals depend on achieving a careful balance between these potentially conflicting goals. When a compromise is possible the PS must appropriately distribute tbc usc of available resources. When a compromise is not possible, then the I'S must Select some of the lowest priority goals for postponement or outright rejection.

The I)S 1PS system can perform on-board all of the functions described above. Goals that can be rejected arc represented in the mission profile as *free tokens*. These arc tokens that have not yet been inserted onto a timeline. Besides expanding the supporting procedure, 1'S has to first decide if tbc goal token will be inserted in tbc appropriate timeline. PS can interleave this decision in the backtracking search procedure described in Section 4 and can therefore explore several goal rejection schemes before returning a final plan. In practice, however, PS dots cannot explore all possible combination of free token achievements but instead follows a statically assigned prioritization scheme (e.g., science goals have highest priority, followed by navigation goals and then by telemetry goals). In the following section we describe examples of goal prioritization due to failures that make certain resources unusable by tbc 1'S.

Goal prioritization schemes make commanding of the spacecraft easier and more robust. It is easier because goal achievement decisions can be postponed to reflect the actual conditions of execution of the plan, making unnecessary extensive contingency analyses in advance; it is more robust because even if ground specifics goals that cannot all be achieved together, the spacecraft will not give up and continue operations by executing a "good enough" commanding sequence.

6. FAILURE RESPONSE

The RA provides two levels of failure responses: an immediate *reactive* response, and a longer term *deliberative* response. This is typical of many autonomy architectures (e.g., Soar [7], Guardian [8]). The reactive behavior provides for fast, real-tinlc responses to failures that could damage the spacecraft if not dealt with immediately like a stuck thruster or a rapidly draining battery. On DS1, the reactive behaviors arc provided by the 1 EXEC and M 1 R. Once the spacecraft is stabilized, the deliberative behavior assesses the impact of the failures on the remaining goals, and determines how to proceed in light of the failures. The deliberative responses arc provided by the planner.

This two level response results in simpler and more robust plans. The plans arc simpler, since they can address only the nominal case and trust that failures will be handled properly as they arise. Failures arc either resolved by the reactive layer and allow tbc plan to continue, or cannot be resolved, in which case the plan breaks and the 1'S generates another nominal plan based on tbc new spacecraft state.

The plans arc also more robust. This is partly due to the failure response mechanism, partly due to the hierarchical nature of tbc RA, and partly due to the plan representation. The hierarchy allows the tokens in the plan correspond to fairly abstract procedures, and the plan representation allows tokens to have flexible start and end times. This allows the EXEC considerable latitude in bow it executes tokens. It can respond to failures by retrying commands, or trying alternate approaches. The extra time needed to respond to the

failures is absorbed by the flexibility in the token's start and end times.

In the following we quickly outline the significance of plans both in the reactive and in the deliberative failure responses.

The Reactive Layer

As mentioned above, the reactive failure responses are handled completely by EXEC and MIR on a token by token basis. During execution, the conditions of validity for each token must be held true for the entire duration of the token. When EXEC executes the procedure corresponding to a token in the plan, it relies on MIR inferring the state of the spacecraft devices based on their telemetry and responses to EXEC commands. If the inferred state is inconsistent with the commanded state, MIR suggests recovery actions to the EXEC. The recovery action must guarantee that the conditions of validity for the token continue to hold.

For example, the MICAS_Turning_On token involves turning on heaters and power switches, taking sonic calibration images, etc. If the MICAS power switch does not transition to the *on* state as commanded, then the MIR may suggest EXEC to re-send the command, and if that fails, to reset the MICAS camera.

Besides asserting validity conditions the plan also allocates resources for each token within which recovery must take place. If this resource allocation is exceeded the token and the plan are declared "broken". For example, the duration associated to each token type in the 1'S model typically has a lower bound (the time needed to execute the token if no failure occurs) and an upper bound (the time needed to execute the maximum allowed number of recovery actions). As a result of the total number of tokens and the topology of the plan, PS may "cat up" some of the duration flexibility, possibly allocating only enough token duration to execute the nominal procedure without recovery. It is also important to notice that given the flexible nature of plans, slack in a plan can be transferred between connected token so that duration flexibility of tokens whose execution succeeds without recovery is made available through constraint propagation to future tokens in case they need it.

The Deliberative Layer- Replanning

1 f the plan breaks, the EXEC gets the spacecraft into a safe con figuration and immediately asks the I'S for a new plan. The I'S assesses the impact of the failure on the mission goals, and attempts to develop a new plan that will achieve them from the current spacecraft state. This is called *replanning*. There is no difference between planning and replanning. The same algorithm is used in both cases. *Replanning* refers to planning after a plan break, whereas *planning* refers to the nominal cycle.

When a replan occurs, the I'S must assess the impact of degraded or unusable devices on the mission goals.[3] For goals generated on-board, the goal generators examine the relevant spacecraft state. If the goals are patently unachievable, no goal is generated. For example, if the IPS engine is declared unusable, then the navigator will not return goals requiring use of the IPS (e.g., 11'S thrusting goals), If the MICAS camera is unavailable, then the same applies to navigation image goals.

[3] The same assessment is also necessary in normal planning conditions. 1 lowever, the devices are fully functional, and the resources are at their expected levels, so there is no impact cm the goals.

After the obviously unachievable on-board goals have been removed by the goal generators themselves, the impact of the failures on the remaining goals is assessed by the planning model. This assessment occurs as a normal part of the planning search.

The planning algorithm attempts to resolve all the open compatibilities in the plan. If one approach fails, it tries alternates until it either generates a plan that satisfies all the compatibilities, or fails.

Knowledge about the impact of the spacecraft state on mission goals is expressed in the compatibilities of the plan model. Each of the goals is represented as a token, and these tokens have compatibilities with tokens on timelines representing device health and spacecraft resources. If the resources are not available, or the device is not sufficiently functional, then the planner will not be able to satisfy the compatibilities on the goal token. If there is some other way to achieve the goal compatibilities that does not require these resources, then the planner will find it and generate a plan accordingly.

If the planner cannot find any way to satisfy the compatibilities, then it will try to reject some of the goals based on a Seal prioritization scheme. 1 f it still cannot find a plan, then the spacecraft remains in standby mode until the ground can intervene during the next DSN pass.

Critical Plans

For critical events, such as encounter, there is no time to recover even at the reactive level, let alone time to recover by replanning. In the DS1 encounter, the MICAS images are spaced so tightly together that an attempt to recover from a failure while taking one image could result in loosing several other images. It is better to simply move on to the next image and hope the fault clears itself.

In traditional missions, these so called *critical sequences* are handled by switching to an alternate fault control mode in which some faults are ignored but critical faults, such as sudden loss of battery power, are still handled.

in DS1, critical sequences are handled within the existing fault control mechanism. The semantics of a few carefully selected tokens are changed such that there is no way they can fail. Specifically, the semantics of take_image are changed to, "attempt to take an image." This token cannot fail, so no recovery actions are needed.

This approach must be used with great caution. All the other tokens in the plan must be consistent with all possible outcomes of the "critical" token. For example, after encounter there are tokens that write the contents of the MICAS image buffer to non-volatile storage. These tokens must not require that the buffer have images in it, since the take_image token does not guarantee that an image will actually be taken. The buffer may well be empty.

7. FORTUITOUS EVENTS

Re-assessments of mission goat achievement by replanning can be also fruitful when spacecraft capabilities are unexpectedly restored or when the spacecraft performs better than expected or when external fortuitous events open the possibility of achieving high-payoff mission goats. From the DS1 planner's perspective these situation are covered by the basic scheme described in the previous sections. The occurrence of advantageous events needs to be detected by EXEC and communicated to the 1'S in the initial state. Other than that, 1'S will perform the same search procedure that already handles the nominal and failure scenarios.

As an example of an unexpectedly restored resource, consider a situation in which 11'S has malfunctioned and EXEC breaks the plan, makes IPS unavailable to the PS and includes in the next telemetry downlink a request for ground to assess the situation. Assume that during the next DSN pass ground is able to run Sonic tests and decides that the malfunction was a fluke and IPS operations can resume. EXEC can now break the plan currently in execution (which did not include tile achievement of "SEP thrust accumulation" goals) and immediately re-invoke the 1'S with an initial state that includes the fact that IPS is now available.

Replanning also allows the spacecraft to take advantage of better-than-expected resource consumption. Consider a two-week plan that allocates three kilograms of fuc] for each week. Assume now that after the first Week, the spacecraft has only utilized two kilograms (the PS may have been conservative in its estimates). If EXEC can independently track actual resource consumption, it can notice the advantageous situation, break the current plan and request a new plan. The additional fuel could be used to achieve additional low-priority goals that had to be rejected in the original plan. This capability will not be explored for DS 1 since EXEC will not independently track resource consumption.

Finally consider a situation in which a spacecraft notices volcanic eruptions on the basis of pictures taken in the early stages of a planetary flyby. This event would be dramatic enough to grant a complete change of the science schedule for the rest of the flyby. Although this capability is not being explored for DS1, it can easily be handled in the current architecture. The science unit would detect tile eruption and request an immediate replan. During the replan, PS asks all the on-board units for their goals. Among these are the new science image goals from the science unit. The I'S can now consider the priorities of the new goals along with the other on-board and mission goals, and decide which ones it will actually achieve.

8. CONCLUSIONS

(In-board planning is a crucial element of spacecraft autonomy. It can reduce mission costs and improve mission quality by allowing high-level commanding, enabling achievement of mission goals in the presence of failures without ground intervention, and taking advantage of fortuitous events.

The 1 DS1 mission marks the first on-board planner to fly on a NASA spacecraft. The validation of this technology will open the way for future autonomous missions.

REFERENCES

[1] Pell, B., Bernard, D., Chien, S., Gat, E., Muscettola, N., Nayak, P., Wagner, M. and Williams, B. 1996. A remote agent prototype for spacecraft autonomy. in *Proceedings of the SPIE Conference on Optical Science, Engineering and Instrumentation*.

[2] Pc]], B., Gat. E., Kessing, R., Muscettola, N., and Smith, B., 1996., Plan Execution for Autonomous Spacecraft in *Proceedings of the AAAI Fall Symposium on Plan Execution* (forthcoming), AAAI Press.

[3] Williams, B.C., and Nayak, 1'.1'., 1996. A mo(icl-based approach to reactive self-configuration s ystems. I n *Proceedings of AAAI-96*, pp 971-978.

[4] Muscettola, N. 1994. 11 S'1'S: Integrating planning and scheduling. in Fox, M., and Zweben, M . , eds, *Intelligent Scheduling*, Morgan Kaufman.

[5] Weld, 1). S., 1994. An Introduction to Least Commitment Planning, *Al Magazine Winter 1094,*

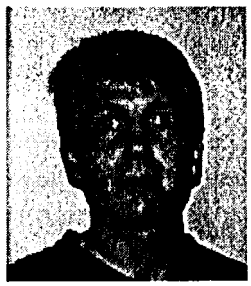
[6] Allen, J.F. and Koomen, J.A. 1983. Planning using a temporal world model. *IJCAI 83. pp.* 741-747.

[7] Tambe, M., Johnson, W.L., Jones, R. M., Koss, F., Laird, J.E., Rosenbloon, P.S., and Schwamb, K. 1995. Intelligent agents for interactive simulation environments. *Al Magazine,*16(1):1 5-39.

[8] Hayes-Roth, B. 1995. An architecture for adaptive intelligent systems. *Artificial Intelligence 72.*

Steve Chien *is Technical Group Supervisor of the Artificial Intelligence Group of the Jet Propulsion Laboratory, California Institute of Technology where he leads efforts in research and development of automated planning and scheduling systems. He is also an adjunct assistant professor in the Department of Computer Science at the University of Southern California. He holds a B.S., M.S., and Ph.D. in Computer Science from the University of Illinois. His research interests are in the areas of: planning and scheduling, operations research, and machine learning.*

Nicola Muscettola *is the lead of the DS1 flight planning team. He also leads at AMES the development of science planners for space-based observatories. He holds a Ph.D. in Computer Science from the Politecnico di Milano, Italy. Research interests span scheduling, planning, temporal data bases and intelligent agents.*



Gregg Rabideau *is a Member of the Technical Staff in the Artificial Intelligence Group at the Jet Propulsion Laboratory, California Institute of Technology. His main focus is in research and development of planning and scheduling systems for automated spacecraft commanding. Other projects include planning and scheduling for the first deep space mission of NASA's New Millennium Program, and for spacecraft design of the Pluto Express project. Gregg holds a B.S. and M.S. degree in Computer Science from the University of Illinois where he specialized in Artificial Intelligence.*

Ben Smith *is a member of the Artificial Intelligence group at JPL, and Deputy Lead of the DS1 planning team at JPL. He holds a Ph.D. in computer science from the University of Southern California. His research interests include intelligent agents, machine learning, and planning.*

David Yan *is a member of the technical staff in the Artificial Intelligence group at the Jet Propulsion Laboratory, California Institute of Technology. He holds a B.S. in Electrical Engineering and Computer Science from the*

University of California at Berkeley. He will be pursuing his M.S. degree in Computer Science at Stanford University in 1996. His research interests include automated planning and scheduling, operating systems, computer architecture and computer networks.

Chuck Fry *and* **Kanna Rajan** *are members of the DS1 planning team at the NASA Ames Research Center, under contract from the Caelum Research Corporation.*